



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Irányítástechnika és Informatika Tanszék

Feladatellenőrző rendszer továbbfejlesztése

SZAKDOLGOZAT

Készítette

Nyíri Tamás

Konzulens

Dr. Szeberényi Imre

2017. december 2.



TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Nyíri Tamás

mérnök informatikus hallgató részére

Feladatellenőrző rendszer továbbfejlesztése

A programozás oktatásában fontos szerepe van az önálló gyakorlásnak, feladatmegoldásnak, amit jelentős mértékben segít a gyors és pontos visszajelzés. Ez gépi eszközökkel jól támogatható, sőt olyan funkciókkal is kiegészíthető, melyek ellenőrzése hagyományos eszközökkel nehézkes. A tanszéken a korábbi szakdolgozatok és diplomatervek keretében elkészített Cporta/Jporta elnevezésű automatikus kiértékelést és ellenőrzést végző feladatkiértékelő rendszer nemcsak a feladatok ellenőrzését segíti, hanem az oktatás egyes adminisztrációs feladatait is egyszerűsíti.

Az újratervezett rendszer felépítésében és kialakításában lényegesen rugalmasabb elődjénél. Így lehetőség nyílik több olyan modul kialakítására is, ami a feladatok megfogalmazását és kiértékelését támogatja. A hallgató feladata a meglévő kiértékelő modulok továbbfejlesztése, új támogató funkciók megvalósítása, jogosultsági rendszer felülvizsgálata, valamint az adminisztrációs funkciók továbbfejlesztése.

Feladatok:

1. Ismerje meg és mutassa be jelenlegi rendszer felépítését és főbb moduljait!
2. Mérje fel és elemezze a rendszerrel kapcsolatos igényeket, melyek oktatói oldalról jelennek meg a feladatok kiírása, ellenőrzése és a tárgyak adminisztrációja terén!
3. Tegyen javaslatot új funkciók megvalósítására (pl. feltöltéskor közvetlen kapcsolat valamilyen verziókezelővel, plágiumkeresés, lefedettség- és egyéb tesztek, tesztek export/import funkciói, kari vezetés által generált újabb adminisztrációs feladatok támogatása, stb.)!
4. Az igények és lehetőségek figyelembevételével egészítse ki a rendszert, úgy hogy minél jobban támogassa az oktatók munkáját.
5. Készítse el a szükséges dokumentációkat!

Tanszéki konzulens: Dr. Szeberényi Imre

Budapest, 2017. október 6.

Dr. Kiss Bálint
egyetemi docens
tanszékvezető

HALLGATÓI NYILATKOZAT

Alulírott *Nyíri Tamás*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózataán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2017. december 2.

Nyíri Tamás

hallgató

Tartalomjegyzék

Bevezető	7
Más automatikus kiértékelő rendszerek	8
Moodle	8
edX, Open edx	8
A szakdolgozat felépítése	9
1. A JPorta	10
1.1. Alapvető felépítés	10
1.2. A rendszer hallgatói oldalról	12
1.3. A rendszer oktatói oldalról	13
1.4. Modulok felépítése	13
2. Hallgatói értékelési rendszer	14
2.1. Dinamikus értékelések	15
2.2. Dinamikus értékelések jelenlegi implementációja	17
2.3. Dinamikus értékelések tesztelése	18
2.4. Dinamikus mezők dinamikus függőségei	19
3. Automatizált feladatkiértékelő modul	23
3.1. Jogosultságkezelés a JPortában	25
3.2. Jogosultságkezelés felülvizsgálata	26
3.3. Kód lefedettség ellenőrzés	28
3.4. Továbbfejlesztési lehetőségek	30
3.4.1. Plágiumkeresés	30
3.4.2. Verziókezelő támogatás	32

Összefoglalás	33
Köszönetnyilvánítás	34
Ábrák jegyzéke	35
Irodalomjegyzék	37

Kivonat

A programozás korszerű oktatásában egyre inkább előtérbe kerül az önálló tanulás és gyakorlás, melynek célja a hallgatók motiválása a mélyebb tudás megszerzésére. Ezt a széleskörben elterjedt internet hozzáférés tette lehetővé, hiszen így bárhol és bármikor elérhetőek ezek a rendszerek mindenki számára.

Ennek támogatására jött létre a BME Irányítástechnika és Informatika Tanszékén először a Cporta, majd később annak utódja, a Jportaként ismert webes oktatást segítő rendszer. A portál feladata egyfelől az oktatók igényeit kielégítő adminisztrációs felület biztosítása, másfelől hogy lehetőséget teremtsen a hallgatók programozási feladatainak automatikus kiértékelésére és ellenőrzésére.

A minél nagyobb fokú megbízható automatizálás mind a hallgatóknak, mind az oktatóknak nagy segítséget biztosít. A hallgatók szinte azonnal értesülnek a feltöltött munkájuk esetleges hiányosságairól, hibáiról, mely megkönnyíti ezek javítását. Oktatói szempontból pedig nagy mértékben csökkenti a feladatok ellenőrzésére fordítandó munkát.

Szakdolgozatomban bemutatom a Jporta meglévő funkcionálisait, különös tekintettel az adminisztrációs részekre és az automatikus kiértékelésre. Megtervezem és implementálom azon funkciókat, melyek akár oktatói, akár hallgatói oldalról növelik a portál értékét. Végezetül pedig javaslatot teszek további fejlesztési lehetőségekre.

Abstract

Bevezető

A programozói ismeretek elsajátításához merőben más módszertanra van szükség, mint egy irodalmi vagy jogász pályán. Az előbbi előnye, hogy megfelelő háttér biztosításával nagyban javítható a tanulási görbe, azonban sajnos az egyetemi körülmények között nincs lehetőség minden halltóval személyesen foglalkozni, hiszen ez óriási többlet munkát róna az oktatókra. Emiatt a hallgatók eredményes, mégis időtakarékos támogatása érdekében egyre nagyobb törekvés indult az automatizálása felé. Ezen megoldások számos előnnyel rendelkezhetnek:

- Az értékeléshez nincs szükség a beadások letöltésére, azok saját számítógépen történő fordítására, futtatására, majd kiértékelésére.
- Azonnali visszajelzés a beadás sikerességéről a hallgatóknak.
- Egyéb beadáshoz kapcsolható metrikákkal is dolgohatunk: futási idő, memóriahasználat, stb.
- Automatikus pontszámítás a számított metrikák alapján.
- Határidők automatikus kezelése.
- Kommunikációs felület a hallgató és oktató között.

Természetesen egy-egy ilyen szoftver elkészítésénél cél az előbbi előnyök legnagyobb mértékű teljesítése, illetve kiegészítése a lokálisan tapasztalt további igényekkel.

Más automatikus kiértékelő rendszerek

Moodle

A Moodle [13] egy teljes körű eLearning rendszer, mely nyílt forráskódú GNU GPL [11] licenc alatt készült. A Moodle-nek mára több, mint 120 ezer felhasználója van, a világ 232 országában [14], ami a nyílt forráskód előnyeivel kombinálva óriási potenciált jelent.

A Moodle általános lehetőséget biztosít az online oktatáshoz. Egyszerűen hozhatunk létre benne kurzusokat, melyekre a jelentkezést akár korlátozhatjuk is. A kurzusokon belül témakörökre bonthatjuk a tananyagot, melynek formája sokféle lehet: pdf fájlok, videók, külső weboldalak, stb. A résztvevők elsajátított tudásának ellenőrzése sem marad el: a Moodle általános rendszert biztosít online tesztek készítésére is, különféle jelleggel, mint pl. több válaszlehetőségből helyes(ek) kiválasztása, egy soros vagy éppen hosszabb saját szavas válaszok. A legtöbb fajta tesztnél lehetőségünk van a helyes válaszok megadására, így a portál azonnal ki is értékeli a beadást, ennek köszönhetően pedig a felhasználó azonnal értesül az elért eredményéről.

Elterjedtségének köszönhetően számos közösségi fejlesztésű modul is készült hozzá, melyek közül találhatunk szép számmal a programozás oktatására fókuszálókat is. Ilyen például a Virtual Programming Lab [17] [7], mely támogat forráskódszerkesztést a böngészőben, programok futtatását, ellenőrzését és plágium ellenőrzést is. Mindezt a felhasználók a már megszokott Moodle környezetben érhetik el a kényelmes használat érdekében.

edX, Open edX

Az edX [9] egy nonprofit online kezdeményezés, melynek alapítói a Harvard University és a Massachusetts Institute of Technology. Ennek segítségével egyetemi szintű kurzusokat tartanak világszerte közel 10 millió felhasználóval és több, mint ezer kurzussal [10].

Az edX rendszer nem csak egyszerű tananyagok megtekintésére biztosít lehetőséget, de videókat és egyéb feladatokat is találunk a kínált kurzusokban. Emellett egyszerűen kapcsolatba léphetünk másokkal, akik a kurzusunkat hallgatják, ha éppen segítségre lenne szükségünk. A kurzusok többnyire ingyenesek, de sok esetben van lehetőségünk fizetés ellenében igazolást kapnunk a sikeresen elvégzett kurzusról.

A portálon lehetőségünk van programozással kapcsolatos kurzusok felvételére is. Ezek keretében pedig online kód írásra, fordításra, futtatásra és a helyesség ellenőrzésére is van mód, amik nélkül az önálló tanulás nagyon nehézkes lenne. Webes kódolást némely kurzusnál Codeboard¹ integrációval van megoldva, melynek köszönhetően nem is kell a tanuláshoz a megfelelő környezeteket telepítenünk a saját gépünkre.

Az Open edX egy nyílt forráskódú platform, melyet az edX fejlesztett ki és tett szabadon hozzáférhetővé saját oktatási környezetek létrehozására. Közel a teljes szerver oldali kód Python nyelven íródott, melynek széleskörű ismertségének (és a rendszer nyílt forráskódú voltának) köszönhetően bárki kiegészítheti a meglévő kódot.

A szakdolgozat felépítése

TODO

¹A Codeboard (<http://codeboard.io>) egy böngésző alapú fejlesztő környezet a programozás oktatás segítésére. Támogatja az edX, moodle, coursera és egyéb platformokat.

1. fejezet

A JPorta

A BME Irányítástechnika és Informatika Tanszékén 2009 és 2015 között aktívan használták a CPorta névre keresztelt automatikus feladatkiértékelő rendszert, melyet a Közigazgatási Informatikai Központ (BME-IK) munkatársai fejlesztettek. Az idő folyamán a rendszert folyamatosan bővítették, azonban egy idő után ez egyre nehezkesebbé vált. Ennek oka a fejlesztőgárda cserélődése, a nem megfelelően átgondolt új funkciók implementálása, illetve az igények változása volt. Ezek miatt 2014-ben új portál fejlesztését kezdték meg a tanszéken [12].

Így született meg a JPorta névre keresztelt oktatás támogató és automatikus feladatkiértékelő rendszer. A rendszer implementálása a Python programnyelv 3-as verziójával valósult meg, ezzel biztosítva a tartós támogatottságot. A webes felület generálására pedig a Django [5] keretrendszer 1.11.5-ös verzióját használja jelenleg, amely ezen felül felelős a belső adatmodellen végzett műveletek adatbázisműveletekre fordításáért és végrehajtásáért is.

1.1. Alapvető felépítés

A JPorta keretein belül a Neptun rendszerhez hasonló tárgy-kurzus struktúra került implementálásra. Ezáltal létrehozhatóak tárgyak, majd ezeken belül további kurzusok a gyakorlatok és a laboratóriumi foglalkozások számára (ld. 1.1. ábra). A tárgyakhoz és a kurzusokhoz külön-külön rendelhetünk oktatókat, hallgatókat viszont csak az utóbbihoz. Az kurzusokhoz rendelt oktatóknak van lehetősége az adott csoport hallgatóit értékelni, jelen-

léteiket adminisztrálni, beadásaikat megtekinteni és körlevelet írni a tagoknak. A tárgyhoz rendelt oktatónak ezen felül joga van minden tárgy szintű adminisztrációhoz is.



1.1. ábra. Tárgy adminisztrációs nézet a kurzusokkal

A kurzusokhoz hozzárendelhetünk különböző számonkérések és jelenlétek eredményét (ld. az 1.2. ábra), ezekből készített dinamikusan számolódó mezőket, illetve automatikusan kiértékelődő feladatokat. Ezekről részletesebben a 2. és a 3. fejezetben lesz szó.

A portálon jelenleg a be nem jelentkezett felhasználóknak nincs elérhető tartalom, így első lépés a kötelező autentikáció. Erre jelenleg két mód áll rendelkezésre:

- BME címtár azonosítás [2]: ez az elsődleges azonosítási forma, ugyanis a célközön-séget képző hallgatók és oktatók mind rendelkeznek ilyen fiókkal. Ennek köszön-hetően egyszerűen, külön regisztráció nélkül hiteles adatokhoz juthatunk.
- Hagyományos felhasználói név és jelszó pár: ez a hitelesítési mód ritkán, csak kivé-teles esetekben használt. Ilyen lehet a portál bemutatására szánt próba felhasználói fiók.

A bejelentkezett felhasználók szerepelhetnek egyes tárgyaknál oktatói, másoknál hall-gatói szerepben. Erre sok esetben szükség is van, hiszen a laboratóriumi foglalkozásokat és gyakorlatokat gyakran felsőbb éves hallgatók tartják. Az oktatók jogairól és egyéb le-hetőségeiről részletesebben az 1.3. fejezetben lesz szó.

Eredményeim korábbi eredményeim

A programozás alapjai 3 - BMEVIIIAB00

L2N ✉

Dátum	Jelenlét	Név	Eredmény	Note
2017-09-05	✓	zh1	2.5	-
2017-09-12	✓	zh2	4	-
2017-09-19	✓	zh3	1.5	-
2017-09-26	📄	zh4	3.5	-
2017-10-03	✓	zh5	3.5	-
2017-10-10	✓	zh6	-	-
2017-10-17	✓	HF spec	-	-
2017-10-24	✓	HF bead	-	-
2017-10-31	✓	Jegy	-	-
2017-11-07	✓	IMSC1	-	-
		beadás hete	-	-

1.2. ábra. Eredmények egy hallgató szemszögéből

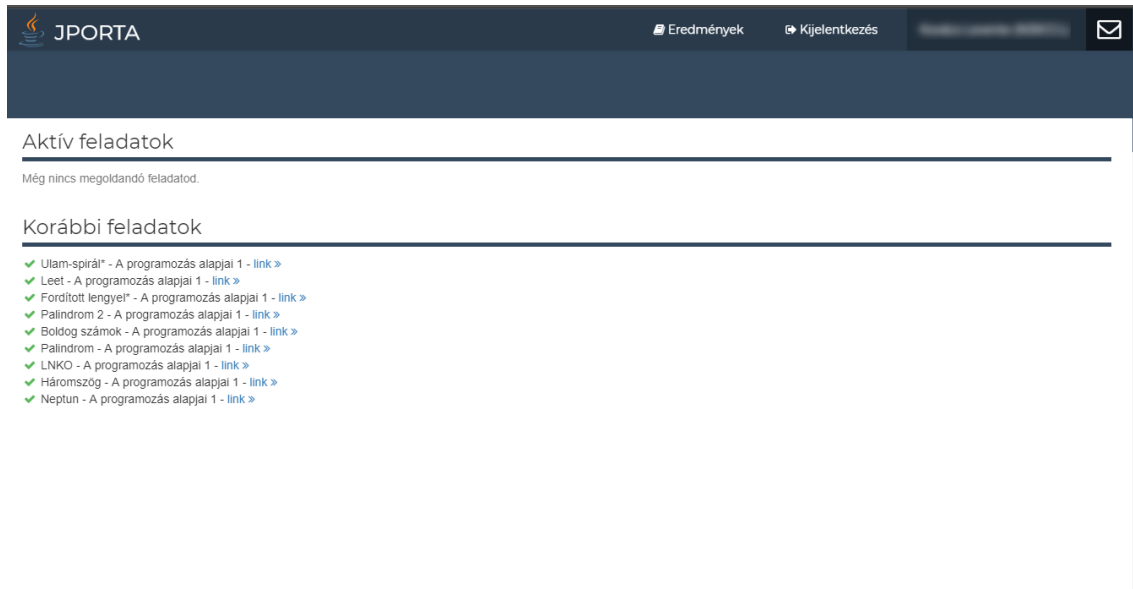
1.2. A rendszer hallgatói oldalról

A portálra belépve a hallgatókat egy összefoglaló nézet fogadja (ld. az 1.3. ábra). Itt találunk egy összefoglalót az aktuális és a már lejárt határidejű online beadandó feladataikról. Egy feladatot kiválasztva pedig lehetőségük nyílik annak beadására (ha a határidő még nem járt le) és az előző beadások részleteinek megtekintésére (ha vannak). Ilyen feladat többféle lehet, gyakran használtak az alábbiak:

- Dokumentum feltöltése: tipikusan házi feladatok specifikációihoz, dokumentációihoz tartozóan, pdf formában.
- Programkód feltöltése: külön állományokban, vagy zip fájlban, mely több forrásfájl is tartalmazhat. Ezeket a rendszer lefordítja, lefuttatja és ellenőrzi a futást, ha az előbbiek sikeresek voltak.

Természetesen nem csak ilyen blokkok létrehozására van lehetősége az oktatóknak. Ezek testreszabhatóságaira a 3. fejezetben fogok részletesen kitérni.

Innen továbbnavigálva a hallgatók megtekinthetik részletesen az aktuális és korábbi félévekben regisztrált eredményeiket, jelenléteiket. A portál ezen felül egyszerűsíti a hallgatók és oktatók közötti kommunikációt üzenetküldési funkcióval, így nincs szükség a



1.3. ábra. Belépő képernyő hallgatói szemszögből

másik fél e-mail címének ismerésére. Minden így küldött üzenetről a címzett automatikus e-mail formájában is értesül, azonban a válasz csak a felületről lehetséges.

1.3. A rendszer oktatói oldalról

Az oktatóknak lehetősége van a tárgyukhoz vagy kurzusukhoz tartozó hallgatók értékelésére, feladatok kiírására és körüzenet küldésére. Az értékelésbe beletartozik a különböző zárthelyi dolgozatok eredményének és az órai jelenlétek vezetése, illetve az automatikusan kiértékelődő feladatok ellenőrzése, esetleges felülbírálni.

A tárgy oktatói tudnak kurzusokat létrehozni, azokba hallgatókat felvenni. Emellett folyamatosan van lehetőségük új értékelések és automatikusan kiértékelődő feladatok hozzáadására. Ezek hozzáadása után a kurzusok oktatói fogják az eredményeket regisztrálni és a beadott feladatokat ellenőrizni, esetlegesen az automatikus értékelést felülbírálni.

1.4. Modulok felépítése

TODO

2. fejezet

Hallgatói értékelési rendszer

A korábban leírtaknak megfelelően a JPorta tárgyaiban lehetőség van különböző értékeléseket felvenni, majd azokat kurzusokhoz rendelni. Ezeket hozzárendelés után a kurzus oktatói tudják értékelni (2.1. ábra).

Új értékelést csak a tárgy adminisztrátorai tudnak létrehozni és kurzusokhoz rendelni. Az ehhez tartozó felület a 2.2. ábrán látható. Minden értékeléshez az alábbi tulajdonságok tartoznak:

- Név: rövid név, mely azonosítja az értékelést, pl. ZH 1
- Típus: előre definiált értékek, melyekhez tartozik egy reguláris kifejezés [16]. Csak olyan értéket vehet fel, ami illeszkedik a hozzá tartozó kifejezésre.
- Súly: meghatározza a sorrendet az értékelések megjelenítésénél.
- Ki értékelheti: kurzusokhoz, vagy csak a tárgyhoz rendelt oktatók értékelhetik.
- Dinamikus-e: az adott értékelés dinamikusan értékelődik-e ki, ld. a 2.1 pontban.
- Privát-e: a privát értékeléseket csak az oktatók látják, a hallgatók nem.
- Megjegyzés: részletes leírása az értékelésnek, tipikusan dinamikus értékelések esetén hasznos.
- Kurzusok: tárgyon belül mely kurzusokhoz akarjuk hozzárendelni az értékelést.

Ezen tulajdonságoknak köszönhetően az értékeléseket egyszerűen és személyreszabhatóan lehet kezelni. Alapvető céljuk a zárthelyi számonkérések eredményének adminisztrá-

L3N Exportál Szerkesztés Törölés Lezárás

Tagok Feladatok Jelenlét Értékelések Üzenetek

Rendezés ↓ név szerint ↓ neptun szerint toggle columns

Név	zh1	zh2	zh3	zh4	zh5	zh6	HF spec	Legjobb 4 átlag >_
[img] [Name]	4	3	2	4	3.5	3	OK	3.62
[img] [Name]	2.5	3.5	0	2	3.5	5	OK	3.62
[img] [Name]	2	2	1	3	1		OK	2.0
[img] [Name]	1.5	5	2	3	2.5	4	OK	3.62
[img] [Name]	1	5	4	5	4.5	3	OK	4.62
[img] [Name]	2.5	4	2.5	3	5	2.5	OK	3.62
[img] [Name]	1.5	3	3.5	0	4		OK	3.0
[img] [Name]	1.5	5	3.5	3	5		OK	4.12
[img] [Name]	5	5	5	3	5	4	OK	5.0

2.1. ábra. Hallgatók értékelései

lása, de bármikor hozzáadhatunk egyéb mezőket is. Ilyen lehet pl. a hallgató házi feladatának személyes bemutatására kijelölt időpont, vagy éppen a házi feladat dokumentáció státusza.

Az univerzalitás egyedüli határa a megfelelő típus megtalálása az értékeléshez, de ez könnyen bővíthető. Új igény felmerülésekor a portál adminisztrátorai hozzáadhatnak új típust, mely egy tetszés szerinti reguláris kifejezésre illeszkedő tartalmat vár.

2.1. Dinamikus értékelések

Az értékelések létrehozásánál hamar felmerült az igény a dinamikus, azaz automatikusan számoló mezők használatára. Ezek nagyban megkönnyítik a félév végi összesítést és a végső jegy meghatározását. Viszont ahhoz, hogy ezt széleskörűen lehessen használni egy teljesen általános rendszert kellett fejleszteni, hiszen minden tárgynak eltérő, akár félévről félévre változó követelményei lehetnek, melyekhez más és más számolásokat, súlyozásokat kellhet végezni.

Ennek megoldására a jelengi implementáció előtt is volt mód, ám az fapadosnak számított. Először az oktatóknak exportálni kellett a meglévő eredményeket egy Excel fájlba, majd ezt a fájlt külső programmal szerkesztve kellett létrehozniuk a dinamikus mezők ér-

+ Új értékelés felvétele

Név*

Result type*

Weight*

Determines the display order of the assessments.

Gradable by*

💡 Százalék sztring: [0-9]{1,2}|100

0-100

💡 NZH jegy sztring: ([0-9]+([\.,][0-9]+)?)|n

Tört pont + 'n'

💡 GO/NOGO sztring: GO|NOGO|go|nogo|igen|nem|IGEN|NEM|I|N|i|n|0|1|OK|NO

go/nogo/igen/nem

💡 Pont (0-10) egész szám: (10|[0-9])

Egész pont 0-10

💡 Tört pont (0-10, 0.5) float: (10([\.,]0)?|[0-9]([\.,][05])?)

Tört pont 0-10 félpontonként

💡 Tört pont float: [0-9]+([\.,][0-9]+)?

Pontozás tizedes ponttal.

💡 Jegy egész szám: [1-5]

1-től 5-ig

2.2. ábra. Értékelés típus hozzáadása

tégeit. Csak ezután tölthették fel a portálra a bővített fájlt, melynek feldolgozása során a JPorta az ebben található értékeket frissítette. Ezt a módszert Kálmán Viktor cserélte le a jelenlegi alternatívára [12].

2.2. Dinamikus értékelések jelenlegi implementációja

Az aktuális megvalósításnak két fő követelményt támasztottak: webes felületen beállíthatónak és az oktatók számára könnyen testreszabhatónak kellett lennie.

Ezek teljesítése érdekében a dinamikus mezők kiszámolására Python nyelven írt kódokat használ a portál. Minden dinamikus mezőhöz webes felületen módosítható kód tartozik, amely megkapja a hallgatókhoz tartozó adatokat, majd ez alapján kiszámolja a mező értékét. A megadott Python kódnak tartalmaznia kell egy *calculate_result* függvényt, mely három szótár (dictionary) típusú paraméterrel rendelkezik:

1. paraméter a jelenléteket tartalmazza külön előadás, laboratóriumi és gyakorlati órákra bontva. Minden típushoz megtalálható, hogy hány alkalommal volt jelen a hallgató (attended), hány alkalommal lett megtagadva a jelenléte (denied), hány alkalommal nem jelent meg (no), illetve hogy összesen hány foglalkozás volt tartva abból a típusból (all).
2. paraméter a (nem dinamikus) értékeléseket tartalmazza, *None* értéket ott, ahol nincs megadva eredmény. A kulcs mindig az adott értékelés azonosítója.
3. paraméterben pedig az automatikusan kiértékelődő feladatok eredményei találhatók. A kulcs itt is mindig az adott feladat azonosítója, értékei pedig az alábbiak:
 - *None*, ha az adott feladatra nem adott be megoldást a hallgató.
 - *0*, ha az adott feladat kiértékelés alatt van.
 - *1*, ha az adott feladat oktatói jóváhagyásra vár.
 - *2*, ha az adott feladat sikertelen.
 - *3*, ha az adott feladat sikeresen lefutott.

Ez a függvény minden hallgatóra egyesével fog meghívódni, visszatérési értéke pedig megadja a mező értékét az adott hallgatónál.

Az alábbi példa kód a Programozás alapjai 3. tárgynál használt egyik dinamikus mezőhöz tartozik, feladata a félév végi átlag kiszámítása. Itt a félév során megírt 6 db zárthelyi dolgozatból a legjobb 4 átlagát kell venni, majd kerekíteni 2 tizedesjegyre. A további kerekítést a laborvezető oktatók végzik.

```
def calculate_result(atds, asms, asgs):
    results = [asms[374], asms[375], asms[376], asms[377], asms[378], asms[379]]
    results = [x for x in results if x is not None]
    results.sort(reverse=True)
    return round(sum(results[:4])/4, 2)
```

Természetesen a kötelező `calculate_result` függvény mellett tetszőleges számban használhatunk segédfüggvényeket is a számolás átláthatósága és egyszerűsítése érdekében.

2.3. Dinamikus értékelések tesztelése

Az így elkészített dinamikus mezőben könnyen előfordulhatnak programozói hibák, emiatt a kód felhasználása előtt elengedhetetlen annak átfogó tesztelése. Szerencsére a korábban elkészült rendszer ezt is támogatja. A 2.3. ábrán látható módon meg tuduk adni teszt bemeneti paramétereket, majd a rendszer (az eredeti kiértékeléssel megegyező módon) kiszámolja a dinamikus mezőhöz tartozó értéket, így meggyőződhetünk a helyes működéséről.

```

1 def calculate_result(atds, asms, asgs):
2     results = [asms[374], asms[375], asms[376], asms[377], asms[378], asms[379]]
3     results = [x for x in results if x is not None]
4     results.sort(reverse=True)
5     return round(sum(results[:4])/4, 2)

```

Visszatérési érték: 0

stdout:

stderr:

[Test function](#)

ID	zh1 (374)	zh2 (375)	zh3 (376)	zh4 (377)	zh5 (378)	zh6 (379)	HF spec (380)	beadás hete (411)	Legjobb 4 átlag
1	<input type="text" value="1"/>	<input type="text" value="2"/>	<input type="text" value="3"/>	<input type="text" value="4"/>	<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text"/>	<input type="text"/>	4.25
2	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text"/>	<input type="text" value="2"/>	<input type="text" value="2"/>	<input type="text"/>	<input type="text"/>	1.5
3	<input type="text" value="0"/>	<input type="text" value="1"/>	<input type="text" value="0"/>	<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text"/>	<input type="text"/>	4
4	<input type="text" value="1"/>	<input type="text" value="4"/>	<input type="text" value="5"/>	<input type="text" value="4"/>	<input type="text" value="4"/>	<input type="text" value="0"/>	<input type="text"/>	<input type="text"/>	4.25

2.3. ábra. Dinamikus mező tesztelése

Az értékek mellett láthatjuk a programunk kimeneteit és visszatérési értékét is. Ezeket akár a teszteléshez is felhasználhatjuk, azonban élesítés előtt célszerű ezeket törölni, mivel a portál minden éles futásról tárolja az ilyen jellegű adatokat az ellenőrizhetőség miatt.

Ugyan a példában nem volt szükség sem a jelenlétekre, se az automatikusan kiértékelődő feladatok eredményére, ezeket a számonkérésekhez hasonlóan fel tudjuk használni, illetve a tesztelés is az előzőekkel megegyezően működik.

2.4. Dinamikus mezők dinamikus függőségei

Ez a rendszer az alapvető igényeket ugyan kielégíti, de hamar felmerült két fő hiányossága:

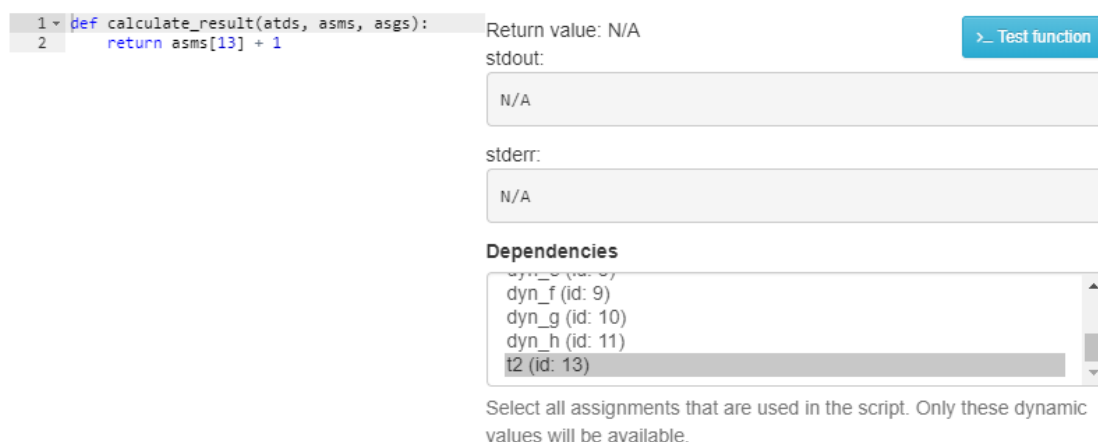
- Dinamikus mezők kiszámítására nem használhatunk más dinamikus mezőket.
- Jelenleg nem lehet egy hallgatóra vagy kurzusra újraszámolni a dinamikus mezők értékeit, hanem mindig a tárgy összes hallgatójára újra kiértékelésre kerülnek ezek. Kezdetben ez nem tűnt nagy pazarlásnak, mivel 10-20 másodpercnél többet nem vesz igénybe a művelet egy 400 fős tárgy esetében [12], azonban felhasználva az előző pont miatt szükséges módosításokat, már sokkal közelebb kerülünk ennek a pontnak a megoldására is.

Az első hiányosság miatt, ha egy tárgy esetében szeretnénk, hogy a hallgatók láthassák külön a zártyhelyi dolgozataik átlagát, majd ezen érték és más mezők alapján egy másik mezőt is hozzá szeretnénk adni, akkor ezt csak kódDuplikációval tehetjük meg. Azaz mindkét mezőnél meg kell adnunk ugyan azt a Python kód részletet, illetve a második mezőnél ezt még ki kell egészítenünk. Ez számos problémát felvet: a kód nem újrahasználható, módosítások hibalehetőséget rejtenek, illetve akár inkonzisztens értékeket is előállítunk, ha a kódokat nem sikerül szinkronban tartani.

A szakdolgozatom keretében erre a problémára is kerestem a megfelelő megoldást, melyet két kulcsfontosságú részre bontottam: függőségek meghatározása, majd ezek alapján a függőségi gráf kiértékelése.

A függőségek meghatározására több módszert is megvizsgáltam. Végül amellet döntöttem, hogy a felhasználóknak kell megadniuk, mely dinamikus mezőket szeretnék használni (ld. 2.4. ábra). Ennek előnye, hogy egyszerű meghatározni a függőségeket (hiszen a felhasználó explicit megadja azokat), hátránya pedig, hogy külön beállítást igényel. Az automatikus függőség felismeréshez szükséges lett volna a mezőhöz tartozó Python kód elemzése, mely értékéhez mérten aránytalanul sok időráfordítást követelt. Emiatt döntöt-

tem a másik megoldás mellett. Ezen információk birtokában pedig bármelyik értékeléshez előállítható a teljes függőségi gráf egy egyszerű rekurzív bejárással.



2.4. ábra. Dinamikus értékelés függőségkezelés

Minden függőségnek jelölt mezőt a többi értékeléssel együtt lehet felhasználni, a 2.2. pontban leírt módon. Így nem kellett módosítani a jelenlegi függvény paraméter listát, illetve a logikailag összetartozó részek hasonlóan érhetőek el.

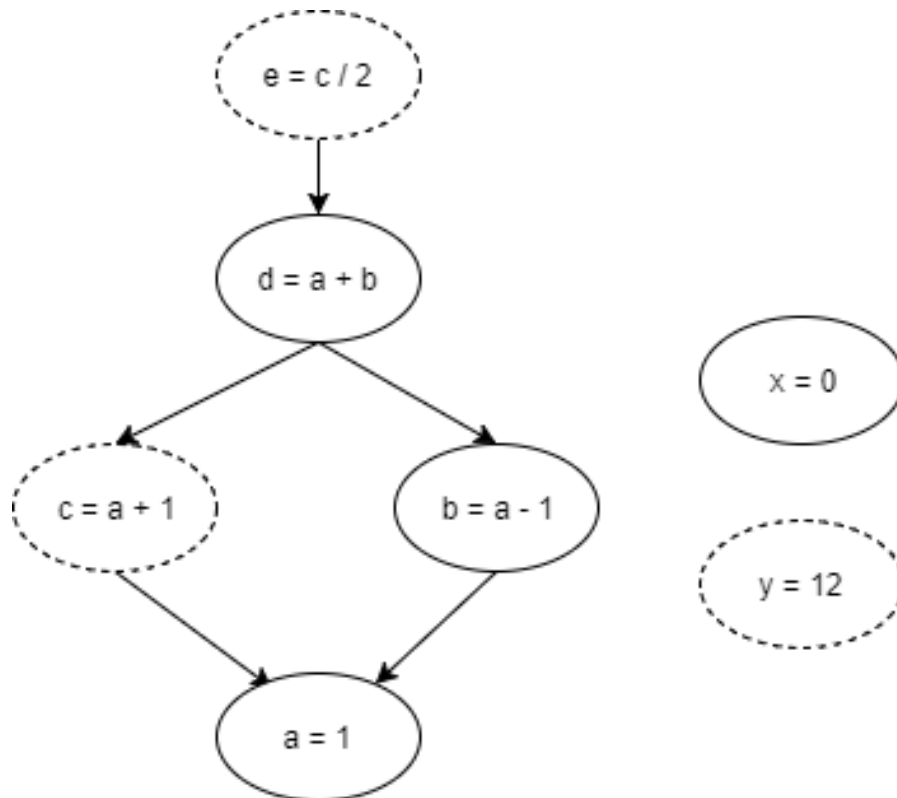
Külön figyelmet kellett fordítani a körkörös függőségek felderítésére, mivel ezek kiértékele végtelen ciklust eredményezne. Annak érdekében, hogy az adatbázisba ne kerülhessen körkörös függőség, ezt még a mentés előtt ellenőrzöm. Megvizsgálom, hogy a megjelölt függőségek közül van-e olyan, ami az aktuálisan módosított példánytól függ. Ha létezik ilyen, akkor ez egy körkörös függőség.

A másik rész a függőségi gráf kiértékelése volt. Dinamikus függőségek nélkül a sorrend nem befolyásolja a végeredményt, azonban ezen megkötések mellett már nem minden sorrend fog helyes eredményt adni.

A 2.5. ábrán láthatunk egy példa függőségi gráfot, ahol

- $a \rightarrow b$ jelenti a -nak b -től való függőségét,
- a folytonos elipszis jelenti, hogy az adott dinamikus értékelés kurzushoz van rendelve, azaz kötelező kiszámolnunk az értékét,
- a szaggatott elipszis jelenti, hogy az értékelés nincs kurzushoz rendelve, azaz az értékét nem kötelező kiszámolni, ha más nem hivatkozik rá.

Mivel a függőségek kialakításánál csak az irányított körmentesség a feltétel, kialakulhatnak különböző összefüggő komponensek és izolált pontok. Ezek közül csak azokat



2.5. ábra. Függőségi gráf

akarjuk kiértékelni, melyekre valóban szükség is van, azaz vagy hozzá van rendelve kurzushoz, vagy valamelyik kurzushoz rendelt mező (közvetlen vagy közvetett) függősége.

Ezek értelmében pl. az a, b, c, d, x kiértékelési sorrend megfelelő eredményt ad, mivel minden elem kiértékelésekor azok függőségeit már kiértékeljük és pontosan azokat a mezőket vettük bele a sorrendbe, amelyekre szükségünk van.

A sorrend előállításához először felépítettem egy függőségi gráfot minden olyan elemről kiindulva, mely kurzushoz van rendelve. Minden így megtalált függőséget hozzáadtam a gráfhoz és rekurzívan megvizsgáltam az ahhoz tartozó függőségeket, ha az még nem létezett a gráfban. Ennek eredményeként előállt a függőségi gráf, mely pontosan a szükséges elemeket tartalmazta.

Ez alapján a sorrend felállításához felhasználtam Dudás Ádám a JPorta más részein is használt ütemezőjét [8]. Ez a megoldás egy függőségi gráf alapján képes megtalálni egy megfelelő sorrendet, sőt a párhuzamosítási lehetőségeket is, de ez utóbbit nem használtam ki.

Ezen kiegészítésekkel elkészült a követelményeknek megfelelő kiegészítés. Továbbfejlesztési lehetőség lehet, hogy a dinamikus függőségek mellett, minden más függőséget is

számon tartsunk. Ezzel minimalizálni lehetne az adatok változása miatt újraszámolandó mezőket, illetve mindig naprakészen lehetne őket tartani.

Emellett segíteni lehetne az oktatók munkáját azzal, ha lennének előre elkészített dinamikus értékelési sablonok. Ezek saját tárgyhoz rendelése után pedig további személyreszabási lehetőségeket tenne lehetővé. Ezzel a rendszer megismerésének folyamata nagyban gyorsítható lenne.

Hallgatói és oktatói oldalról nézve is hiányzik, hogy a számolás alatt álló mezők a korábbi értéküket tartják meg, így inkonzisztens állapotot mutatnak. Erre az időre célszerű lenne invalidálni a korábbi értékeket és jelölni, hogy még ezek az adatok nem állnak rendelkezésre.

3. fejezet

Automatizált feladatkiértékelő modul

A JPorta egyik fő funkciója a beadott feladatmegoldások automatikus kiértékelése. Ennek tervezése során törekedtek az általános, könnyen bővíthető megoldás megtalálására, amikkel akár nem programozási jellegű feladatokat is ki lehet adni. [8]

Az elkészült modul blokkokat használ alapvető építő elemeinek, melyek egy-egy kiértékelési feladatot valósítanak meg. Ennek köszönhetően új igény esetén nem kell a meglévő blokkokat módosítani, csak az új funkciót megvalósító blokkot implementálni.

A blokkok közötti kommunikációt bemeneti és kimeneti csatlakozóik teszik lehetővé. Ezek szabadon összeköthetők bármely más blokk ellenkező típusú csatlakozójával, pl. a felhasználói fájl blokk kimenetét ráköthetjük a GCC fordító blokk bemenetére, így lefordíthatjuk az adott forrásfájlt (rendelkezésre álló blokkok pontos leírását ld. 3.1). Ezen kívül lehetnek olyan paraméterei is egy blokknak, melyeket adminisztrátori felületen kell beállítani, pl. fordítás esetén a fordítónak átadott kapcsolók.

A beadás végső eredménye egy speciális blokk értéke lesz. Ennek a SubmissionResult blokknak egy bemente van, mely ha igaz a beadás sikeresnek tekinthető, ellenkező esetben sikertelen.

A blokkok kiértékelése (hasonlóan a 2.4. pontban leírtakhoz) egy függőségi gráf felépítésével kezdődik, melynek kezdőpontja a fentebb említett SubmissionResult blokk. Csak azok a blokkok kerülnek kiértékelésre, amelyek (közvetve vagy közvetlen) függnek ettől a bloktól, hiszen a többi nem befolyásolja a beadás sikerességét. Itt sem megengedettek a körkörös függőségek, tehát az elkészült gráfnak körmentesnek kell lennie.

Blokk név	Leírás	Bemenetek	Kimenetek
Specifikáció	Feladat pontos leírása	Sablon paraméterek	-
Szkript	Egyedi paraméterek generálási módja	Alapértelmezett bemenet, bemeneti fájlok, környezeti változók	Alapértelmezett (hiba) kimenet, kimeneti fájlok
Szerzői fájl	Feladat kiíró által feltöltött fájl	-	Fájl
Felhasználói fájl	Hallgató által feltöltött fájl	-	Fájl
GCC fordító (3.1. ábra).	C és C++ fordító	Archívumok, bemeneti fájlok, könyvtárak, forrásfájlok, bemeneti paraméterek	Kimeneti fájlok, futtatható állomány, alapértelmezett (hiba) kimenet
Futtató	Program futtató	Alapértelmezett bemenet, környezeti változók, archívumok, bemeneti fájlok	Alapértelmezett (hiba) kimenet, kimeneti fájlok
Szkript ellenőrző	Bash/Python szkripttel a bemenet ellenőrzése	Bármely blokk kimenete	Alapértelmezett (hiba) kimenet, eredmény
Szövege ellenőrző	Bemenet összehasonlítása konstans szöveggel	Szöveges elvárt szöveg, valódi szöveg paraméterei	Eredmény
Oktatói jóváhagyás	Oktató kézi ellenőrzés	Függőségek	Eredmény
Logikai ÉS	Blokkok kimeneteinek és kapcsolata	Argumentumok	Eredmény
Logikai VAGY	Blokkok kimeneteinek vagy kapcsolata	Argumentumok	Eredmény

3.1. táblázat. *Rendelkezésre álló blokkok*

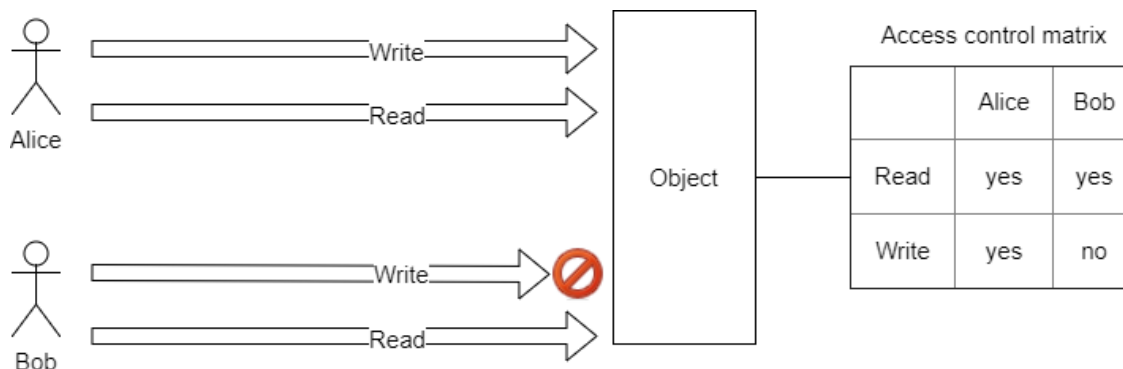
3.1. ábra. GCC fordító blokk adminisztrációs felülete

3.1. Jogosultságkezelés a JPortában

A JPorta által használt Django keretrendszer beépítetten tartalmaz egy egyszerű jogosultság kezelő rendszert. Ez lehetővé teszi különböző jogosultságok hozzárendelését felhasználókhöz, vagy felhasználói csoportokhoz. Minden Django modellhez [3] tartozik alapértelmezés szerint három jogosultság, melyeket a keretrendszer automatikusan hoz létre. Ezek a létrehozás (add), módosítás (change) és törlés (delete) jogosultságok. Sok esetben már ezek is elegendő lehetnek számunkra, de bármikor létrehozhatunk új jogosultságokat, melyekkel személyre szabottabban tudjuk kezelni a hozzáférést egy-egy funkcióhoz. Ezeknek köszönhetően alapvetően egyszerűen kezelhetjük a jogosultságok kérdését [6]. Fontos megjegyezni, hogy Django-ban a *superuser*-nek jelölt felhasználók automatikusan rendelkeznek minden jogosultsággal.

A JPorta ezen felül egyedi megoldást alkalmaz az egyes objektum példányok elérésének szabályozására, mely az access control list (ACL, [15]) technikához hasonló. Az ACL minden objektumhoz rendel egy mátrixot, mely oszlopaiban a személyek (actor), soraiban pedig az egyes műveletek (action) találhatóak. Egy személy akkor hajthat végre egy műveletet, ha a mátrix általa és a művelet által meghatározott cellája igaz értéket tartalmaz.

A 3.2. ábra egy ilyen példát szemléltet: az objektumhoz tartozó hozzáférési mátrix értelmében Bob-nak csak olvasási, Alice-nak pedig olvasási és írási joga van. Emiatt Alice mind az olvasási, mind az írási műveletet sikeresen végrehajthatja, Bob azonban írási műveletet nem hajthat végre.



3.2. ábra. ACL működése és egy hozzáférési mátrix

A portál közvetlen jogosultságok helyett jogosultsági szinteket használ, tipikusan tulajdonosi (owner) és operátori (operator) szintekkel, de ez modellenként eltérő lehet. Ilyenkor ha egy felhasználót egy objektum tulajdonosának jelölünk, egyben operátori jogosultságokkal is felruházzuk. Ha egy felhasználónak nem kívánunk jogosultságot adni az adott objektumhoz, akkor nem jelöljük egyik szinten sem. Fontos megjegyezni, hogy a *superuser*-nek jelölt felhasználók itt is automatikusan rendelkeznek minden jogosultsági szinttel.

Ez a megoldás kifinomultabbnak mondható az eredeti ACL technikához képest, mivel így különböző műveletek halmazát rendelhetjük egy adott szinthez, pl. az operátor jogosult megtekinteni és módosítani az adott objektum tulajdonságait, de a törléshez már tulajdonosi szintre van szükség.

3.2. Jogosultságkezelés felülvizsgálata

Az automatikus kiértékelő modulnál a jogosultság kezelés korábban nem készült el megfelelően, így ugyan hallgatói oldalról nézve minden jól működött, a feladatokat csak *superuser* jogokkal rendelkező adminisztrátorok hozhatták létre. Ez pedig nagyban megnehezítette felsőbbéves hallgatók, laborvezetők bevonását a rendelkezésre álló feladatok bővítésére. Ennek oka, hogy ekkor minden más adatot is láttak volna a portálon, beleértve minden hallgató eredményeit, megoldásait, ami nem megengedhető.

Ezek értelmében a jogosultság kezelés kibővítésével a cél az volt, hogy egyszerűen és biztonságosan lehessen automatikusan kiértékelődő feladatok létrehozására és szerkesztésére. Az így elkészült rendszer alkalmazza mindkét fenti hozzáférés szabályozási módszert a maximális biztonság és testreszabottság elérésének érdekében.

Ennek elkészítéséhez először a Django jogosultságok ellenőrzését implementáltam, melyhez a beépített *PermissionRequiredMixin* osztályt használtam. Ez egyszerűen teszi lehetővé a jogosultságok meglétének ellenőrzését, csak meg kell adnunk a szükséges jogosultság(ok) listáját, a többit pedig a keretrendszerre bízhatjuk. [4]

Két jogosultságot használtam fel:

- *view_exercise*: meglétével a felhasználó megtekintheti az összes eddig létrehozott automatikusan kiértékelődő feladatot (exercise). Ennek köszönhetően láthatja, ha már létezik hasonló feladat, mint amire szüksége van, illetve ötletet meríthet a többi feladatból. Emellett pedig a tanulási folyamatot is segíti, hiszen a már meglévő feladatok megértésével fény derülhet az addig rejtélyesnek tűnő működésre.
- *create_exercise*: meglétével a felhasználó létrehozhat új feladatokat. Ilyenkor lehetősége van új, üres feladatot létrehozni vagy egy korábbiról másolatot készíteni.

Ezek a jogosultságok fedik le általánosságban a hozzáférések szabályozását. A konkrét feladat példányok hozzáféréseinek konrtollálásához az előzőek kiegészítésére operátori és tulajdonosi szintet használtam. Ha egy felhasználóhoz egyik sincs hozzárendelve, akkor csak megtekintheti az adott feladatot, nem módosíthatja annak semmilyen elemét. A felhasználói szintek beállítása a 3.3. ábrán látható.

- Operátor (operator): az adott feladat példány alap adatait (cím, leírás és címkék), meglévő blokkjait, azok összeköttetéseit tudja módosítani. Lehetősége van más operátor szintű felhasználók felvételére is, illetve egy beadott feladatra újra lefuttatni a kiértékelési folyamatot.
- Tulajdonos (owner): az adott feladat példányhoz teljes jogkörrel rendelkezik. Az operátori szint jogain felül hozzáadhat, törölhet blokkokat, más személyeket tulajdonosi szintre emelhet és akár törölheti is a feladatot. Csak tulajdonosok publikálhatnak vagy vonhatnak vissza feladatokat¹.

Végezetül a szerzői fájlok (ld. a 3. fejezet) hozzáféréseit ellenőriztem. Itt van lehetőségünk a hallgatók elől elrejteni az adott fájlt, ugyanakkor ez csak a feladatbeadásnál való listázottságát módosítja csak. A fájlok webcím alapján közvetlenül továbbra is elérhetőek

¹A feladat publikálása késznek jelöli azt, melyet csak ilyen állapotban lehet kurzushoz rendelni.

The exercise is published, modifying existing blocks or adding new ones is not possible in this state.




Áttekintés

Blocks

+ Blokk hozzáadása

✗ Failed evaluations

Access

Felhasználó	Level	✗
 [User Name]	Operator	<input type="checkbox"/>
 [User Name]	Operator	<input type="checkbox"/>
 [User Name]	Tulajdonos	<input type="checkbox"/>
+ Felhasználónév	Operator	

Mentés

3.3. ábra. Feladat példányhoz tartozó jogosultsági szintek beállítása

maradtak, így próbálgatással az összes titkosnak hitt szerzői fájl tartalmát le lehetett kérdezni. Ennek megoldása egyszerűnek bizonyult, csak módosítani kellett a kérést kezelő függvényben a felhasználó és a visszaadandó fájl ellenőrzését. Eddig a fájl titkos voltától függetlenül visszaadásra került a tartalom.

3.3. Kód lefedettség ellenőrzés

A JPorta elődje révén felmerült az igény a C++ projektek esetében a kód lefedettség ellenőrzésére is. A rendszer bővíthető felépítettsége miatt a kiegészítés a meglévő blokkok és struktúrák lényeges módosítása nélkül lehetséges. Ennek implementálása az alábbi lépéseket foglalta magában:

1. Különféle lefedettség ellenőrzők vizsgálata, a megfelelő kiválasztása.
2. Szükséges Django modellek módosításainak tervezése és implementálása.
3. Feladat blokk tervezése és implementálása az adminisztrációs felületre.
4. Kapott nyers adatok feldolgozása.

5. Felhasználó (hallgató) számára megjelenítés elkészítése.

Megvizsgáltam különböző C++ programozási nyelvet támogató kód lefedettség ellenőrző eszközöket, melyek közül a *gcov* program használata mellett döntöttem. Ennek kimenete kiegészíti a forráskódot azzal, hogy mely sorai hányszor kerültek futtatásra. Minden sor elején szerepel, hogy hányszor futott le. Ezt egy kettőspont követi, majd a sor száma az eredeti forrásfájl szerint, végül pedig a hozzá tartozó kód. Ha az adott sor nem futott le a tesztesetek során, akkor az elején szám helyett ##### vagy ==== szerepel. Ennek futtatásához szükséges egyfelől a fordításkor megfelelő kapcsoló átadása, illetve a fordításkor és futtatáskor keletkezett *gcno* és *gcda* fájlok továbbítása a *gcov* számára. Tehát a keletkezett fájl formátuma az alábbihoz hasonló:

```
-:      1:#include <iostream>
-:      2:
#####:  3:void do_not_run_me()
-:      4:{
#####:  5:      std::cout << "do_not_run_me" << std::endl;
#####:  6:}
-:      7:
1:      8:void run_me()
-:      9:{
1:     10:      std::cout << "do_not_run_me" << std::endl;
1:     11:}
-:     12:
1:     13:int main()
1:     14:{
1:     15:      run_me();
1:     16:}
```

Az elkészült blokk bemenetei a fordítási egységenként keletkezett *gcno* és *gcda* fájlok és a forrásfájlok. Beállítási lehetőségként szerepel a minimálisan elfogadott lefedettségi érték, mely alatt a beadás sikertelennek bizonyul. Ezután elkészítettem a szükséges adminisztrátori felületet a blokkhoz (ld. 3.4. ábra). Ennek bal oldali oszlopában találhatóak a bemenetek bekötésére szolgáló vezérlők (melyek más blokkok kimenetei), középen a blokk tulajdonságainak beállítása, végül jobb oldalon a kimenet definiálása, mely más blokk bemenetére köthető.

A nyers adatok feldolgozását a modell osztály valósítja meg. Ez a szöveget a megadott formátum szerint elemzi, majd elkészíti a megjelenítéshez szükséges adatstruktúrát.

Végül elkészítettem a hallgatói felületen látható grafikus megjelenítést (ld. 3.5. és 3.6. ábra). Itt a hallgató fájlokra bontba megtekintheti, mely sorokat érintette legalább egyszer

3.4. ábra. Kódlefedettség ellenőrző blokk

a tesztesetek során (ezek zöld háttérrel vannak jelölve), illetve azokat, amelyeket egyszer sem (ezek piros háttérrel). A nem színezett sorokban nem található végrehajtható kód. Emellett megtalálható a fájlankénti és az összesített kódlefedettség is százalékos formában.

✓ Check #2

Overall code coverage: 77.28 %

Details

string2.h	File coverage: 100%
Rekord.h	File coverage: 0%
Uniq.h	File coverage: 0%
Settings.h	File coverage: 100%
main.cpp	File coverage: 85.34%
Rekord.cpp	File coverage: 82.50%
RekordListElem.cpp	File coverage: 84.62%
Uniq.cpp	File coverage: 100%

3.5. ábra. Lefedettség ellenőrző hallgatói oldalról

3.4. Továbbfejlesztési lehetőségek

3.4.1. Plágiumkeresés

A portálon beadott megoldásoknak csak akkor van a tanulás szempontjából vett értéke, ha azokat az adott hallgató meg nem engedett segédeszközök nélkül, önállóan oldotta

38.	{
39.	i = false;
40.	}
41.	else if(strcmp(args[ij], "-f") == 0)
42.	{
43.	f = atoi(args[++ij]);
44.	}
45.	else if(strstr(args[ij], "--skip-fields=") != NULL) //--skip-fields=f
46.	{
47.	args[ij] += 14;
48.	f = atoi(args[ij]);
49.	args[ij] -= 14;
50.	}
51.	else if(strcmp(args[ij], "-s") == 0)
52.	{
53.	s = atoi(args[++ij]);
54.	}

3.6. ábra. Lefedettség ellenőrző hallgatói oldalról

meg. Internetes beadások lévén ennek teljeskörű ellenőrzése önmagában is nehezen vagy egyáltalán nem megoldható feladat.

Elégséges megoldásként célszerű lenne implementálni egy plágium² ellenőrző blokkot. Itt a blokk bemenete lehet a beadáshoz tartozó forrásfájlok összessége, fő paramétere egy százalékos érték, melyet meghaladó hasonlóság esetén a beadást plágiumgyanúsak jelöljük, kimenete pedig azon beadások halmaza, amelyekhez az adott forrásfájlok hasonlóságot mutattak.

Írott szövegnél történő plagizálás felderítésénél is felmerül az átfogalmazás problémája. Azaz hogyan detektáljuk, ha a plagizáló nem szó szerint vett át adott részeket, hanem módosítva, de jelentésüket mégis megtartva. Programozási feladatok esetében is felmerül ez a probléma, hiszen sokféle olyan módosítást eszközölhetünk az eredeti forrásfájlon (annak részletes megértése nélkül), mely így látszólag már különbözik. Erre néhány példa lehet a változók, osztályok, struktúrák átnevezése, sorrendjük megváltoztatása, kommentek elhelyezése vagy törlése a kódból, stb.

A plágium detektálást nehezítő faktorok miatt felmerül más szolgáltatások használata is erre a célra. Ilyen szolgáltatás többek között az amerikai Stanford egyetemen készült Moss (Measure Of Software Similarity, [1]) szoftver is, mely képes a feltöltött forráskódok között elvégezni az összehasonlítást. Jelenleg többek között támogatott programozási nyelvek a

²Plágium: „szellemi tolvajlás, más művének közlése saját név alatt, a mű alapgondolatának vagy részeitének felhasználása a szerzőre való hivatkozás nélkül” (Magyar Értelmező Szótár)

C, C++, Java, JavaScript, C#, Python, így tehát az oktatásban használt nyelvek túlnyomó többsége.

Másik problémaként felmerül, hogy a hallgatók nem csak egymás beadásaiból másolhatnak részeket, hanem az interneten keringő egyre több nyílt forráskódú program bármelyikéből. Ezek felismeréséhez számon további probléma megoldására lenne szükség.

3.4.2. Verziókezelő támogatás

Összefoglalás

Köszönetnyilvánítás

Ábrák jegyzéke

1.1. Tárgy adminisztrációs nézet a kurzusokkal	11
1.2. Eredmények egy hallgató szemszögéből	12
1.3. Belépő képernyő hallgatói szemszögéből	13
2.1. Hallgatók értékelései	15
2.2. Értékelés típus hozzáadása	16
2.3. Dinamikus mező tesztelése	18
2.4. Dinamikus értékelés függőségkezelés	20
2.5. Függőségi gráf	21
3.1. GCC fordító blokk adminisztrációs felülete	25
3.2. ACL működése és egy hozzáférési mátrix	26
3.3. Feladat példányhoz tartozó jogosultsági szintek beállítása	28
3.4. Kódlefedettség ellenőrző blokk	30
3.5. Lefedettség ellenőrző hallgatói oldalról	30
3.6. Lefedettség ellenőrző hallgatói oldalról	31

Irodalomjegyzék

- [1] Alex Aiken: A System for Detecting Software Similarity. <https://theory.stanford.edu/~aiken/moss/>. Elérve: 2017. november 30.
- [2] Budapesti Műszaki és Gazdaságtudományi Egyetem: A címtárról röviden. <https://login.bme.hu/>. Elérve: 2017. november 14.
- [3] Django Software Foundation: Models. <https://docs.djangoproject.com/en/1.11/topics/db/models/>. Elérve: 2017. november 28.
- [4] Django Software Foundation: The PermissionRequiredMixin mixin. <https://docs.djangoproject.com/en/1.11/topics/auth/default/#the-permissionrequiredmixin-mixin>. Elérve: 2017. november 28.
- [5] Django Software Foundation: The web framework for perfectionists with deadlines. <https://www.djangoproject.com/>. Elérve: 2017. november 14.
- [6] Django Software Foundation: Using the Django authentication system. <https://docs.djangoproject.com/en/1.11/topics/auth/default/#permissions-and-authorization>. Elérve: 2017. november 28.
- [7] Dominique Thiébaud: Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in. *Journal of Computing Sciences in Colleges*, 30. évf. (June 2015) 6. sz., 145–151. p.
- [8] Dudás Ádám: Feladatlíró modul online oktatás rendszerhez. Diplomaterv (Budapesti Műszaki és Gazdaságtudományi Egyetem). 2016. december.
- [9] edX Inc.: About EdX. <https://www.edx.org/about-us>. Elérve: 2017. november 11.

- [10] edX Inc.: Year in Review: edX in 2016. <https://blog.edx.org/year-review-edx-2016?track=blog>. Elérve: 2017. november 11.
- [11] Free Software Foundation, Inc.: GNU General Public Licenc. <https://www.gnu.org/licenses/gpl.html>. Elérve: 2017. november 09.
- [12] Kálmán Viktor: Integrált feladatellenőrző és oktatórendszer. Diplomaterv (Budapesti Műszaki és Gazdaságtudományi Egyetem). 2016. december.
- [13] Moodle Pty Ltd: About moodle. https://docs.moodle.org/33/en/About_Moodle. Elérve: 2017. november 09.
- [14] Moodle Pty Ltd: Moodle Statistics. <https://moodle.net/stats/>. Elérve: 2017. november 09.
- [15] R. Shirey - Network Working Group: RCF 4949. <https://tools.ietf.org/html/rfc4949>. Elérve: 2017. november 29.
- [16] The IEEE and The Open Group: Regular Expressions. http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html. Elérve: 2017. november 18.
- [17] Virtual Programming Lab. <http://vpl.dis.ulpgc.es/>. Elérve: 2017. november 09.